# Exercise 2 – Implementing Dependency Injection

## 1. Objective

To implement Dependency Injection (DI) in a Spring-based Library Management Application using Spring's Inversion of Control (IoC) container via XML configuration.

## 2. Problem Statement / Scenario

We aim to establish proper dependency management between the BookService and BookRepository classes. The goal is to inject BookRepository into BookService using setter injection defined in the applicationContext.xml.

## 3. Implementation Steps

### Step 1: XML Configuration (applicationContext.xml)

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <!-- BookRepository Bean -->  
 <bean id="bookRepository" class="com.example.repository.BookRepository" />  
  
 <!-- BookService Bean with Setter Injection -->  
 <bean id="bookService" class="com.example.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

### Step 2: Update BookService Class

package com.example.service;  
  
import com.example.repository.BookRepository;  
  
public class BookService {  
  
 private BookRepository bookRepository;  
  
 // Setter for Dependency Injection  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void listBooks() {  
 bookRepository.fetchAllBooks();  
 }  
}

### Step 3: Test the Configuration (LibraryManagementApplication.java)

package com.example;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import com.example.service.BookService;  
  
public class LibraryManagementApplication {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService = (BookService) context.getBean("bookService");  
  
 // Test the DI by calling a method  
 bookService.listBooks();  
 }  
}

## 4. Output Verification

[INFO] Spring Context Initialized Successfully.  
[INFO] BookRepository: Fetching all books from database...

## 5. Conclusion

Using Spring's IoC container and DI, we cleanly separated responsibilities and achieved loose coupling between components. This setup ensures easier testing, better maintainability, and flexible configuration in our Library Management System.